(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ENHANCEMENT OF IMAGE PROCESSING CAPABILITIES FOR DIFFERENT ENVIRONMENTS

by

Erkan Aykaç

June, 1991

Thesis Advisor: Charles W. Therrien

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): EC	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
ENHANCEMENT OF IMAGE PROCESSING CAPABILITIES FOR DIFFERENT ENVIRONMENTS

12. PERSONAL AUTHOR(S): Erkan Aykaç

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM TO	1991, June	49

16. SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Digital Image Processing

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis provides a set of tools for the enhancement of digital image processing capabilities on MS-DOS, UNIX, and VM operating systems with computer programming languages APL, C, FORTRAN, and MATLAB. The tools consist of input/output functions for images, programs for displaying images on SUN SPARCstations, and IBM-PC compatibles, and finally "toolkit" for MATLAB to implement the basic digital image processing functions.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Charles W. Therrien	22b. TELEPHONE (Include Area Code): (408) 646-3347; 22c. OFFICE SYMBOL: EC/Ti

Approved for public release; distribution is unlimited

Enhancement of Image Processing Capabilities for Different Environments

by

Erkan Aykaç
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1985

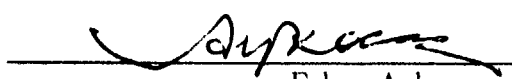
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN
ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

June 1991

Author: 
Erkan Aykaç

Approved by: 
Charles W. Therrien, Thesis Advisor


Jeffrey B. Burl, Second Reader


Michael A. Morgan, Chairman
Electrical and Computer Engineering

ABSTRACT

This thesis provides a set of tools for the enhancement of digital image processing capabilities on MS-DOS, UNIX, and VM operating systems with computer programming languages APL, C, FORTRAN, and MATLAB. The tools consist of input/output functions for images, programs for displaying images on SUN SPARCstations and IBM-PC compatibles and finally a "toolkit" for MATLAB to implement the basic digital image processing functions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. AN OVERVIEW AND PURPOSE

Image Processing has a wide range of application areas like enhancement of images obtained via satellites and object identification for military, medical and other purposes. Because vision is probably the most important and most used of the five human senses, we always like to use images in our presentations. Since image data is important to us, we want to store image data in a reliable, easy, and flexible way. We need consistent and flexible file formats that will allow us to use the same data in different environments quite easily.

Once we have stored the images, then we want to manipulate them, alter some properties of them, and store again in the same format. Thus we need convenient Input/Output procedures from within computer programs. A further requirement is that the images be accessible from a variety of programming environments. In other words, we want to have simple functions or subroutines to read and write the images from within the following languages or environments:

- APL
- C
- FORTRAN
- MATLAB

and perhaps others. Further we desire to have this capability available in several operating system environments, so the user of say MATLAB or C will need to use only a single procedure regardless of the operating system he or she is using. These

operating systems include MS-DOS on IBM-PC compatible microprocessors, UNIX on SUN workstations and the VAX, and VM on the mainframe. In other words we want to present a common interface. In this thesis, we have written such subroutines and functions for all of the above languages and provided them in both the DOS and the UNIX environment. These functions can also be easily ported to VM if desired.

Displaying image data is another separate and important part of digital image processing. Since we want to display image data in different environments, we need to have different programs for each different system due to different display capabilities and display drivers for specific systems. The display functions should be easy, flexible, intelligent, and provide a consistent interface to the user as much as possible. In this thesis, we have provided display programs for both MS-DOS (PC) and UNIX (SUN Workstation) environments to display images. The use of the display functions in both environments is similar.

Processing image data with a computer program is easy but processing image data with MATLAB is fun! Since MATLAB is flexible, user friendly, and interactive, it provides an extremely efficient way to process image data and develop new algorithms. A very basic image processing "toolkit" was written as a part of this thesis and is available now for use with MATLAB. The combination of this image processing toolkit with I/O and display functions, which also operate under MATLAB, provides a very powerful environment in which to perform image processing and develop new algorithms. This environment is, as stated before, available both on PC's and on SUN workstations.

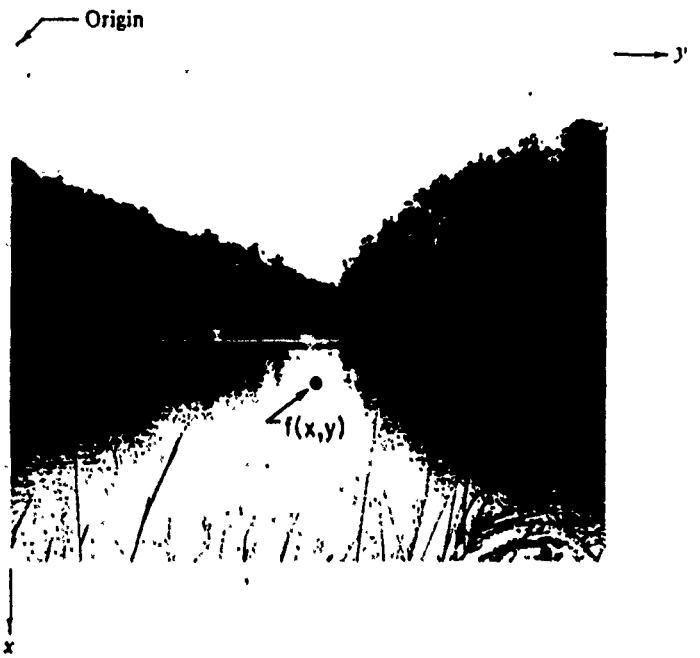


Figure 1.1: Digital Image Representation

B. DATA I/O

We will use the digital image representation presented in Gonzalez [Ref. 1]. The image is represented in a modified Cartesian coordinate system with origin in the upper left corner of the image. The coordinate system is depicted in Figure 1.1.

The smallest spatially discretized image element is called a pixel; at each pixel we have an intensity value representing the gray level from dark to full bright. The data can be thought of as arranged in a matrix. We will restrict ourselves to an 8 bit representation of intensities with values ranging from **0 (darkest)** to **255 (brightest)**. In the case of color images, we have three 8-bit intensity values for red, green, and blue components of the composite real color. These are represented as three separate files on disk or as matrices within a program. Spatial sampling of an image depends on the resolution of the digitizer used; the most common image

sizes are powers of two such as 512 x 512 or 256 x 256. In the latter example, it takes 256 x 256 = 65,536 bytes to represent the image. Image data stored in this form is called **raw data**. We may either store the image data in **raw data form**, or we may add some more information to explain what the image is, when it was captured, the size of the image, and other things. The extra information is put into **a header record** and added to the image data file. Frequently, we want to be able to read/save both raw data images (without header) and images with a header. We have provided easy ways to input and output image data in different environments and programming languages as mentioned earlier.

C. DISPLAY ROUTINES

Display of the image data will be on PC's and SUN Workstations. Both systems are flexible and easy to use. We have developed a complete capability in both environments for monochrome images. In the case of color images, we have some limitations. On the PC, these limitations are due to the restricted quantity of colors for a given palette with the VGA mode. Since SUN Workstations handle displaying color images in a different way than PC's, we cannot display color images in raw data form. If three data files for red, green and blue intensities can be converted into **rasterfile** form however, then we can display them in color. Rasterfiles are defined in the Sun PixRect Reference Manual [Ref. 2].

D. MATLAB TOOLKIT

Several simple image processing tools have been developed. Since notation and definition becomes important in defining and expressing the underlying mathematics, we tried to follow the conventions used in Gonzalez [Ref. 1]. The basic toolkit has functions to perform lowpass filtering, highpass filtering, edge detection (4 types), histogram specification, and image size reduction.

II. DATA MANAGEMENT

A. DATA FILE STRUCTURES

Image data files we will be working with are of two kinds, namely raw data form and raw data with a header. Both of these are binary files, so you cannot edit them using a text editor. The raw data form is standard in image processing and is the type provided on tape by USC/SIPI (University of Southern California/Signal and Image Processing Institute) and other sources. The image is stored in a raster scan order; the first byte of the file corresponds to the intensity at the origin, the second byte to the one on the right of origin, and so on (See Figure 1.1).

Headers for image files in general follow no standard conventions. Our header format was adopted from the ITEX/PC format [Ref. 3]. This has the advantage that images acquired using the PC image processing station in Spanagel 315 can be read directly in all other environments and operating systems addressed in this thesis. The first two bytes of the header consists of the letters 'I' and 'M' to distinguish it from other types of files. This is followed by the length of the comment area, horizontal size of the image, vertical size of the image, the horizontal coordinate of the image on the screen, the vertical coordinate of the image on the screen, format of the image file, a reserved area, and comments. The structure of the header is detailed in Table 2.1 below.

B. I/O ROUTINES FOR MATLAB

1. PC-MATLAB ROUTINES

For PC-MATLAB, the input and output routines and utilities are **readim**, **rim**, **readheader**, **saveim**, and **putim**. These functions are MEX files written in

TABLE 2.1: IMAGE FILE ORGANIZATION

	Byte Location in the File	Purpose of the Bytes
	0 - 1	Letters 'I' and 'M' to distinguish ITEX/PC images from the others. (Hex 49, 4D)
H	2 - 3	Size of comment area, low byte and high byte. These two bytes determine the length of comments which start at byte 64.
E	4 - 5	Horizontal size of image in pixels, 4 is low byte and 5 is high byte (to be multiplied by 256)
A	6 - 7	Vertical size of image in pixels (6 low, 7 high)
D	8 - 9	Horizontal screen coordinates for origin (not necessarily 0)
E	10 - 11	Vertical screen coordinates for origin (10 low, 11 high)
R	12 - 13	Format of image file; possible values are: 0:8 bits/pixel, 1:Compressed data
	14 - 63	Reserved for future use
	64 - xx	Comment in ASCII. Variable size, but the length is specified in bytes 2-3.
D A T A	xx + 1 - END	Data follows the comment. Data length should be product of sizes specified in bytes 4-5 and 6-7. Data is stored in rasterscan form.

the C language (Microsoft C 5.0) and then compiled with the MEX utility that comes with PC-MATLAB. The PC-MATLAB manual explains how to create MEX files on pages 43-58 [Ref. 4]. More information about MEX files can be found in Appendix A. All of these functions have corresponding m-files (same function names) which have only comments in them, to explain how to use the function, and which is typed to the user when the help facility in MATLAB is used.

Function **readheader** reads the header of the image data file, if it exists, and displays header information for the image. If the header does not exist, **readheader** lets us know that the image is in raw-data form and gives its estimated size (as a square image).

Function **readim** reads the **entire** image data file and creates a matrix variable in MATLAB. Since PC-MATLAB limits the maximum number of elements that a matrix can have to 8188 (which corresponds to a 90 x 90 square image), it should always be used with great care. If we limit ourselves to square images that are powers of 2, the largest image that we can read into PC-MATLAB is 64 x 64. This limitation applies only to PC-MATLAB; for 386 MATLAB and PRO-MATLAB, the matrix size is unlimited. This means we can work with images of sizes 1024 x 1024 or even larger. Functions for 386 MATLAB and PRO-MATLAB are discussed in the following two sections. It is useful to check the image file with **readheader** first to determine its size before using **readim**.

Function **rim** is more flexible than readim. It allows us to read portions of an image. While using **rim**, one should specify the coordinates of the leftmost corner, sub-image sizes, and also the filename.

Some common features of both **rim**, and **readim** are:

- If the filename is given without any extension, the default ".img" is added; however, any extension given overrides the default.

- The functions automatically sense the file type and skip the header part (if it exists) and return data into a matrix variable as mentioned above.

- If the image size is not as large as requested, an error message is displayed while trying to read.

Function **saveim** saves a matrix variable in MATLAB to a binary data file **with header**. This function needs an input matrix, a comment for the header (optional), and an MS-DOS filename. If the extension is omitted, the default ".img" is added.

Function **putim** saves an image matrix in MATLAB as a binary file in **raw-data** form. As in the case of **saveim**, it needs an input matrix, and a DOS filename (no comments are possible). In this thesis, we have been working with color USC/SIPI images and when we needed a monochrome USC/SIPI image, we used the .red component of the color image like a monochrome image. So the default extension is ".red" when **putim** is used.

The above routines are in the form of MEX files for PC-MATLAB which were created using Microsoft C 5.0 [Ref. 5] and the MEX utilities supplied with PC-MATLAB. Since these functions are compiled executable codes, they are fast and convenient ways to import/export image data to PC-MATLAB. These are the key functions that make it possible to do image processing within MATLAB. A summary of I/O functions is given in Table 2.1 below; their usage is further explained in the related m-files, which are accessed with the help command in MATLAB.

2. 386 MATLAB Routines

Data I/O Functions for 386 MATLAB are the same as for PC-MATLAB. The advantage of 386 MATLAB is the unlimited size of the matrix variables which allows us to read the entire image, of size 512 x 512 as an example, into a matrix.

TABLE 2.2: SUMMARY OF MATLAB I/O FUNCTIONS

Function Name and Usage	Description
readheader (*'filename'*)	Input is *filename*; displays header information, and estimated image size.
x = readim (*'filename'*);	Input is *filename*, output is variable x; reads the **entire** image.
x = rim (*'filename'*, *xo*, *yo*, *dx*, *dy*);	Inputs are *filename*, origin coordinates *xo*, *yo*, and sub-image sizes *dx*, *dy*. Reads a **portion** of image.
saveim (x, *'filename'*, *'comment'*)	Inputs are image matrix x, *filename*, and *comment* (optional). Saves to a binary MS-DOS file **with header.**
putim (x, *'filename'*)	Inputs are image matrix x, *filename*. Saves to a binary MS-DOS file **without header.**

Also, speed of processing is increased with 386 MATLAB. The I/O functions for 386 MATLAB are created using Metaware High C and the MEX utilities that are supplied with 386 MATLAB. These routines are functionally equivalent to the PC-MATLAB routines.

3. PRO-MATLAB Routines

For PRO-MATLAB, we have functions **readim**, **rim**, **readheader**, **putim**, and **savim** exactly like the ones in the other two environments. They are used in exactly the same way and with the same conventions as in Table 2.2. These functions were created using the C compiler on the SUN workstations [Ref. 6] and the PRO-MATLAB CMEX utility. There are no restrictions on the size of images that can be handled other than availability of storage on the workstation.

C. I/O ROUTINES FOR APL

The APL functions READIH, READIM, and SAVEIM are equivalent to the functions readheader, readim, and saveim in the MATLAB environment. The functions are implemented under IBM APL2 for the mainframe and for the PC [Ref. 7]. Their use is given in Table 2.3.

D. I/O ROUTINES FOR C

The I/O routines for use with C consist of two object modules **getsize** and **getimage** to read images and **saveimage** to write image files. These object modules are ready to be linked with any C program. The function names should be declared as externals and unsigned char pointers should be defined for the image data to be read. The use of these object modules are clarified with two example C programs **readtest** and **writest**. The source code for these two programs is shown below in Figures 2.1 and 2.2.

TABLE 2.3: APL I/O FUNCTIONS

Function Name & Use	Description
H ← READIM '*filename*'	Reads the contents of the header of image *filename* into variable H in the workspace.
M ← READIM '*filename*'	Reads image data into variable M. Shape of M is determined by information in the header.
M SAVEIM '*filename*'	Saves array M as an image file. Automatically creates header with dimensional information.

```
Jun 13 11:30 1991  creadtest.c Page 1


#include <stdio.h>
#include <malloc.h>

extern void   getsize();
extern void   getimage();

main()
{
        long            total_size;
        int              i;
        char             fname[48];
        unsigned char    *datas;   /* define the ptr to get data */

        fprintf(stdout,"enter filename : \n");
        scanf("%s",fname);

        /* get size of image */
        getsize(fname,&total_size);
        fprintf(stdout,"total size : %ld \n",total_size);

        /* allocate memory */
        if((datas=(unsigned char *) malloc(total_size)) == NULL ){
                fprintf(stderr,"memory is not enough\n");
                exit(-1);
        }

        /* read image datas */
        getimage(fname,datas);

        for(i=0;i<9;i++)
                fprintf(stderr,"data[%d]=%x\n",i,datas[i]);

        free(datas);

}
```

Figure 2.1: Creadtest.c Program

```
Jun 12 15:17 1991  cwritetest.c Page 1

#include <stdio.h>

extern void     saveimage();

main()
{
        int     i;
        char    *fname="data2";
        unsigned char   datas[9];

        for(i=0;i<9;i++)
                datas[i]= i;
        saveimage(fname,datas,9);

}
```

Figure 2.2: Cwritest.c Program

The code for **readtest** shows a typical use of **getsize** and **getimage**. Alternatively, one may run readheader first to get the image size and define arrays in his/her program instead of using pointers. However, use of C with the dynamic memory allocation provision permits reading image data before its size is known. In C the file I/O is byte oriented and data can be read one byte at a time, which is the most natural way to do it.

Some special things to note about the sample read program are:

- Include files <stdio.h> and <malloc.h>;
- Definition of external functions **getsize**() and get **image**();
- Definition of variables for image size, input filename, and pointer to data;
- Use of **getsize**();
- Dynamic memory allocation for data;
- Reading of data with **getimage**();
- Processing of data in the program; (This is not included in the test program)
- Freeing of allocated memory.

Some special things to notice about the **writest** program are:

- Include file <stdio.h>;
- Definition of **saveimage**() as an external function;
- Definition of variables for output filenames;
- Call of function **saveimage**().

A limitation exists with the use of these modules on PC's to images less than 256 x 256 in size. Since the Intel 8086 based architecture addresses memory in 64k segments, each data block is limited by most C-compilers to 64k bytes. For the case of the VAX and SUN workstations, there is no restriction on image size.

E. I/O ROUTINES FOR FORTRAN

From the user's point of view, the FORTRAN input/output subroutines are very similar to the C subroutines. They were written with the FORTRAN compiler on SUN workstations [Ref. 8]. The two subroutines **writeimage** and **readimage** are used to save an image data array to a file and read an image file into an array respectively. It is easier to copy the source code than to try to link the object files, because these subroutines are very short. One difference from the C programs is, since the FORTRAN file I/O is record oriented, the file is read as blocks instead of bytes. This detail is actually hidden from the programmer, since the subroutine **readimage** takes care of reading blocks and simply returns an array of data to the calling program. Since the record size is **fixed**, it only works for headerless files. Two test FORTRAN programs **freadtest** and **fwritest** illustrate the details of using these subroutines.

A summary of I/O functions for all different environments is shown in Table 2.4.

TABLE 2.4: SUMMARY OF I/O FUNCTIONS

Operation Performed	MATLAB	APL	C	FORTRAN
reading header information	readheader	READIH	readheader	—
reading image data	readim rim	READIM	getsize getimage	readimage
saving image data	saveim putim	SAVEIM	saveimage	saveimage

III. DISPLAY ROUTINES

A. PC DISPLAY ROUTINE

If we process an image, displaying or printing (using a high resolution printer) is essential in order to compare the processed image with the image before processing. After we examine these images, we can further process the original image to get better results until we reach a desired threshold. Displaying images on IBM PC's is done via DISPLAY.EXE. The source code was written using Microsoft Quick C 2.0. DISPLAY requires a VGA card and monitor, and needs some space on hard (or floppy) disk to make working (temporary) files. The exact space needed depends on the image file(s) to be displayed. For monochrome images, DISPLAY needs a space as large as the image file. For color display, it needs a space three times larger than one of the components of the colored image (since three files are manipulated). In case of insufficient disk space, DOS will terminate DISPLAY and report the error.

DISPLAY is quite flexible and a user with just a little knowledge of DOS can use DISPLAY quite easily. Typing "DISPLAY" at the DOS prompt will cause it to prompt for a filename. Filenames may be entered with or without an extension. Filenames without an extension are first assumed to refer to a color image and extensions .RED, .GRN, and .BLU are appended to the original filename. If these color image files do not exist, then the extension .IMG is appended to the original filename and DISPLAY tries to find a monochrome image with this filename. If the image files are not in the current directory, but in a different directory, then complete pathnames should be given. DISPLAY determines whether or not the image file has a header or consists of just raw-data, and displays it automatically. It is possible

to display a component of a color image as a (black and white) monochrome image. This is done by giving the filename with the extension (e.g., lenna.red).

DISPLAY contains a menu to display left, center, or right portions of an image. This is needed for images that are greater than 320 x 200 pixels in size. Display uses 320 x 200 pixels with 256 colors in VGA. Color images do not look as natural because of coarse quantization of the color palette in VGA. If the image is greater than the screen size, it can be moved with arrow keys to see the hidden portions. The escape key takes you back to the menu. It is possible to reduce the image size, using the reduce option from the menu, by factors of two, three, or four. Since most images are square images, color or monochrome images without a header are assumed to be square. Image files with a header may be of any size, and not necessarily square.

DISPLAY also has UNIX-like features. On the DOS command line, it is possible to give switches and a filename or a complete pathname simultaneously. This results in direct display, skipping the menu. The possible command line switches are -c, -l, -r for center, left, and right portions of the image respectively. A detailed usage of DISPLAY.EXE can be found in READ.DIS, a portion of which is shown below:

NAME

DISPLAY [option . . .] [filename]

OPTIONS

-c Display center part of the image.

-l Display left part of the image.

-r Display right part of the image.

B. UNIX DISPLAY ROUTINE

1. Command Line Mode

Display in UNIX is done via the **show** command.[1] The current version is implemented on SUN SPARCstations. It requires a color or gray scale monitor with 8 bit frame buffer. **Show** works within SUNVIEW, the windowing environment for SUN workstations. Typing **show** on the command line will cause it to prompt for a filename and start in interactive mode. After providing the filename, it will ask for the image size, but will provide a default value, so that you may just hit return rather than giving a size. It will then ask for the displayed image size, which may be smaller (or at most equal to) the stored image size. Again it has a default, so to see the entire image you just hit return. It further prompts for the bits/pixel (default is 8 bits/pixel) and a scale factor. The default scale factor is 1; negative values reduce the display size.

Show can also be used in a noninteractive mode, by supplying the filename with **show** directly on the command line. This will cause **show** to use all default values without any prompt. **Show** can also display images in **rasterfile** form as described in the Pixrect Reference Manual of SUN workstations [Ref.2]. A minor bug with **show** is that it does not display unless you move the pointer into the image window using the mouse.

A very useful feature provided by **show** is that you can find the intensity of a pixel by moving the pointer with the mouse and then holding down the right mouse button. The coordinates (X,Y) and the intensity at this pixel (Z) will be displayed, and an option to quit the image window also is provided at this point.

[1]The current version of this program was obtained courtesy of the University of Southern California/Signal and Image Processing Institute.

2. Display From Within PRO-MATLAB

Although both PC-MATLAB and PRO-MATLAB can use operating system commands using the "!" escape operator, the PC DISPLAY function does not work from within PC MATLAB. In the UNIX environment, however, we can use **show** within Pro MATLAB using the "!" operator of MATLAB. We have provided a script file **show.m** that will display image **matrices** directly from within **PRO-MATLAB**. This m-file takes a matrix as its argument and opens a temporary window to display the image matrix. The elements of the matrix should be integers between 0-255. Interactive use of **show** can also be activated from within PRO-MATLAB by typing "!show". **Show** will then prompt for the image name (which in this case must be a file) and the other parameters remain exactly as before.

IV. MATLAB TOOLKIT

A basic set of image processing functions were also written for use with MATLAB. We refer to these functions as the image processing "toolkit".

A. DISCUSSION OF FUNCTIONS

Functions for basic image processing tasks provided in the toolkit are **equalize, gradienth, gradientv, highpass, hisplot, histogram, laplacian, lowpass, reduce,** and **sobel.** All these functions are m-files written as they are described in Gonzalez [Ref. 1].

Histogram can be used in two ways. The first usage plots the histogram for a given image matrix. The second usage does not plot the histogram of the given image matrix, but instead returns a vector that has data ready for plotting by the function **hisplot. Hisplot** is especially useful for fast display of a known histogram. **Histogram** takes a short time to make the necessary calculations but after it is done, by assigning the result to a variable, we can save computation time for the second and following displays of the same plot.

Hisplot plots data returned by the function **histogram.** The abscissa is intensity level from 0 to 255 and the ordinate is the density of the corresponding intensity.

Equalize can also be used in two ways. In the first usage, it produces the usual histogram equalization, based on an approximately uniform output histogram. In the second usage, we can specify the desired histogram of the output image. For direct histogram specification, **equalize** needs a look-up table as its input as well as the image matrix. The look-up table is a vector of length 256 where the values

in this vector map their positional values to new intensities. If for instance, the 5th element of the look-up table (vector) is 127, then all intensities that have the value 4 in the original image will be replaced by 127.

The **highpass** function is a circularly symmetric ideal highpass filter. Frequencies in the region that is outside the circle are passed with unit gain and spatial frequencies that lie within the circle are blocked.

Lowpass executes an ideal circularly symmetric lowpass filter in the spatial frequency domain. In contrast to the **highpass** function, the spatial frequency components that lie within a circle of the given radius are passed while components outside the circle are blocked.

For edge detection, there are four algorithms, namely **gradienth, gradientv, sobel,** and **laplacian. Gradienth** detects horizontal edges in the image, while **gradientv** detects vertical edges in the image. **Sobel** is a combination of **gradienth** and **gradientv** to detect both horizontal and vertical edges. **Laplacian** makes use of a different mask in which the corners of the mask are zeros. This makes it quite sensitive to the noise in the images. The masks for **gradienth, gradientv,** and **laplacian** are shown in Figure 4.1.

Reduce is useful for compressing large images. **Reduce** converts an image to a desired smaller size by using neighborhood averaging techniques. A reduction size of 2 will reduce a 512 x 512 image to a 256 x 256 image. The basic image processing functions in the toolkit are summarized in Table 4.1.

B. EXAMPLES USING FUNCTIONS

One of the USC/SIPI images "lenna" of size 512 x 512 (color image) has been used to illustrate the basic functions. The red component is used to make comparisons in gray scale, more easily. To make calculations faster, the function

TABLE 4.1: BASIC IMAGE PROCESSING FUNCTIONS

Function and Use	Operation or Effect on Image
histogram (x) or y = histogram (x);	If no output variable (y) exists then plots histogram, otherwise returns a vector y for IIISPLOT.
hisplot(y)	Plots intensity density vector returned by histogram.
y = equalize (x): or y = equalize (x,t);	In the first usage, histogram equalization is performed. If an additional look-up table t is supplied, then direct histogram equalization is performed.
y = highpass (x,r);	Ideal highpass filter with radius r.
y = lowpass (x,r);	Ideal lowpass filter with radius r.
y = gradienth (x);	Gradient horizontal edge detector.
y = gradientv (z);	Gradient vertical edge detector.
y = sobel (z);	SOBEL omnidirectional edge detector.
y = laplacian (x);	Laplacian omnidirectional edge detector.
y = reduce (x,t);	Reduces image size by k with neighborhood averaging.

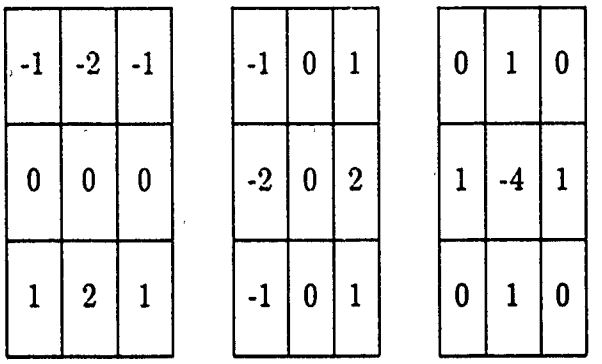


Figure 4.1: Masks Used for Edge Detection: (a) Gradienth, (b) Gradientv, (c) Laplacian

reduce is used first to reduce the image size to 256 x 256 from 512 x 512. Then all the other functions are applied to this reduced image. Figure 4.2 shows the original (512 x 512) image "lenna.red". Figure 4.3 shows the reduced image (256 x 256) using the **reduce** function. Figure 4.4 shows the reduced image on the top left; the equalized image on the top-right; on the bottom a direct histogram specification was used to get the negative of the input image. The look-up table contains the values 0 to 255 in reverse order. Figure 4.5 shows the histogram for the reduced image on the top, and the histogram for the equalized image on the bottom. Figure 4.6 shows the specified look-up table and histogram of the output image. Note how the values are the reverse of those in Figure 4.4(a). Figure 4.7 shows the application of lowpass and highpass filtering. The first row shows **lowpass** filtered images with radii 30 pixels and 80 pixels from left to right. The second row shows **highpass** filtered images with radii 5 pixels and 10 pixels from left to right. Figure 4.8 illustrates the application of edge detectors. The first row shows **sobel** and **gradienth** applied from left to right. The second row shows **gradientv** and **laplacian** applied from left to right.

Figure 4.2: Lenna.red

Figure 4.3: Reduced Image

Figure 4.4: Histogram Equalized Images

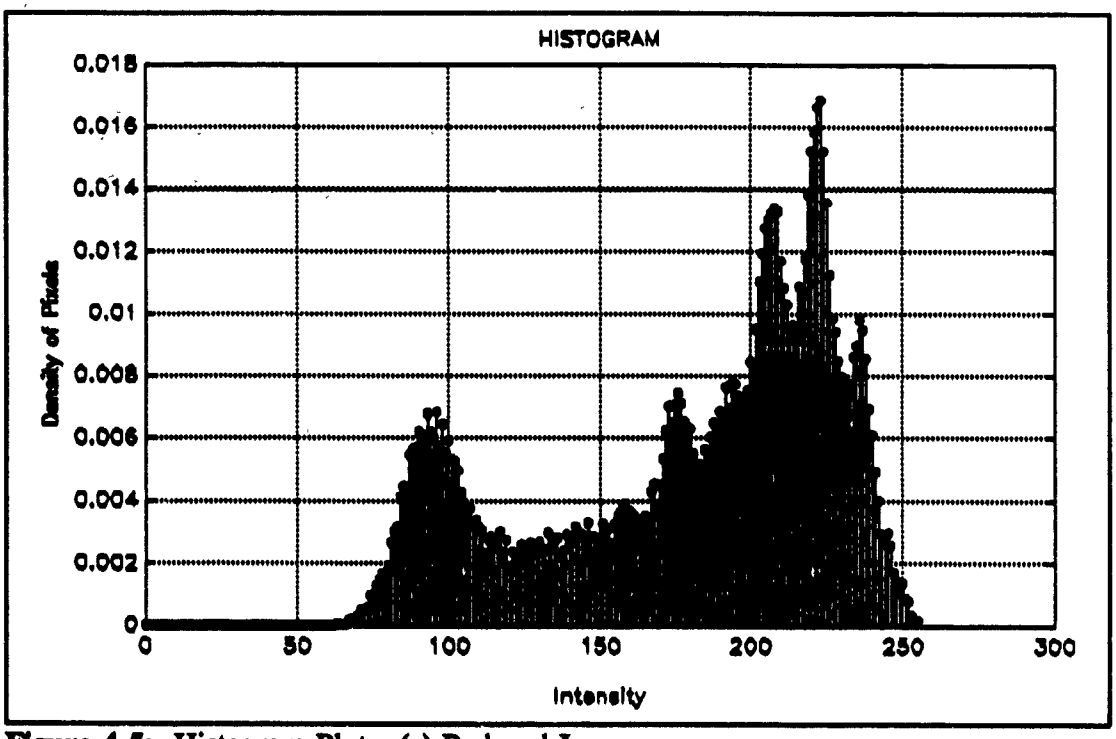


Figure 4.5: Histogram Plots: (a) Reduced Image.

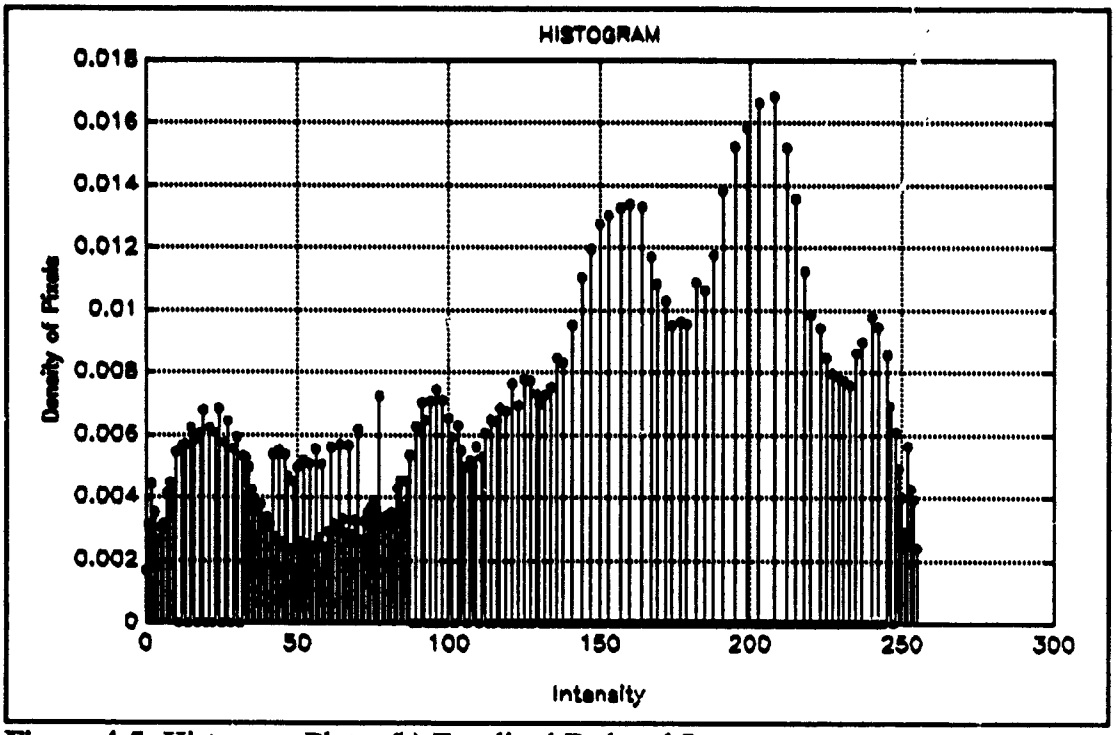


Figure 4.5: Histogram Plots: (b) Equalized Reduced Image.

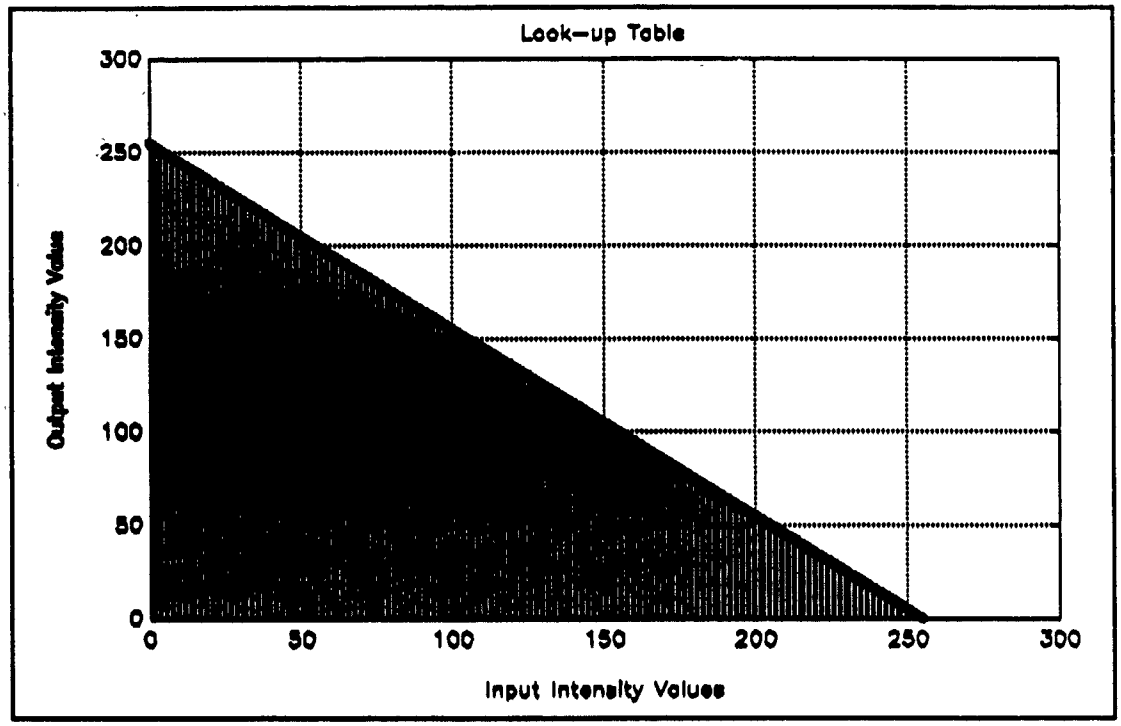


Figure 4.6: Direct Histogram Specification: (a) Look-up Table.

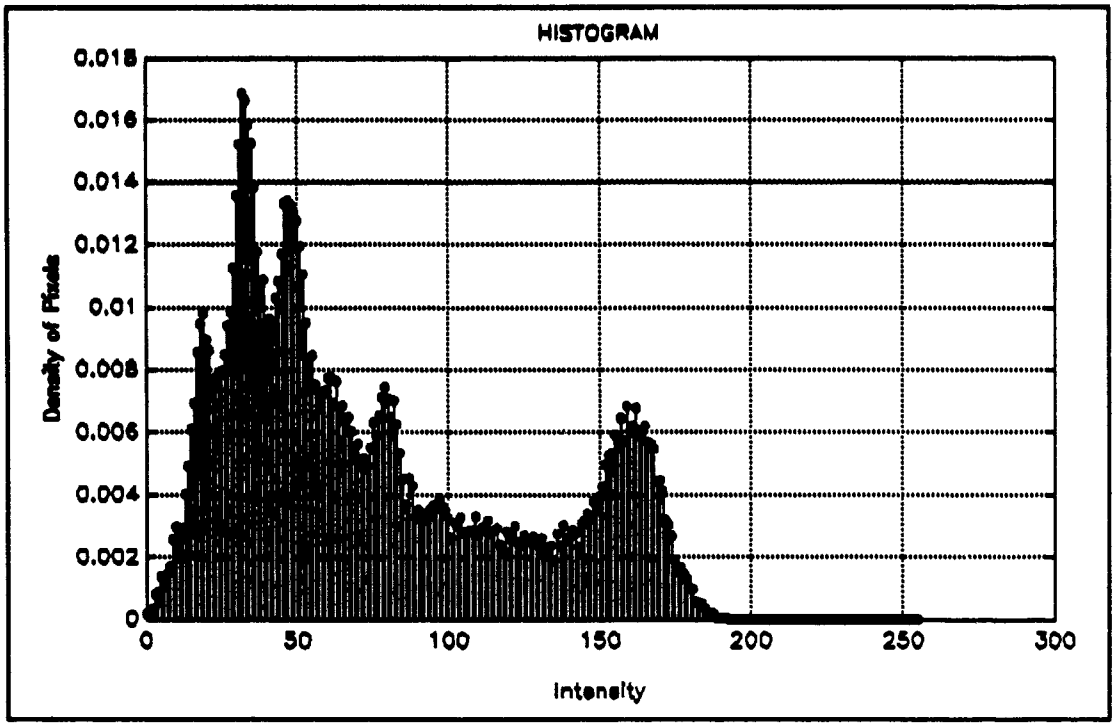


Figure 4.6: Direct Histogram Specification: (b) Histogram of Output Image.

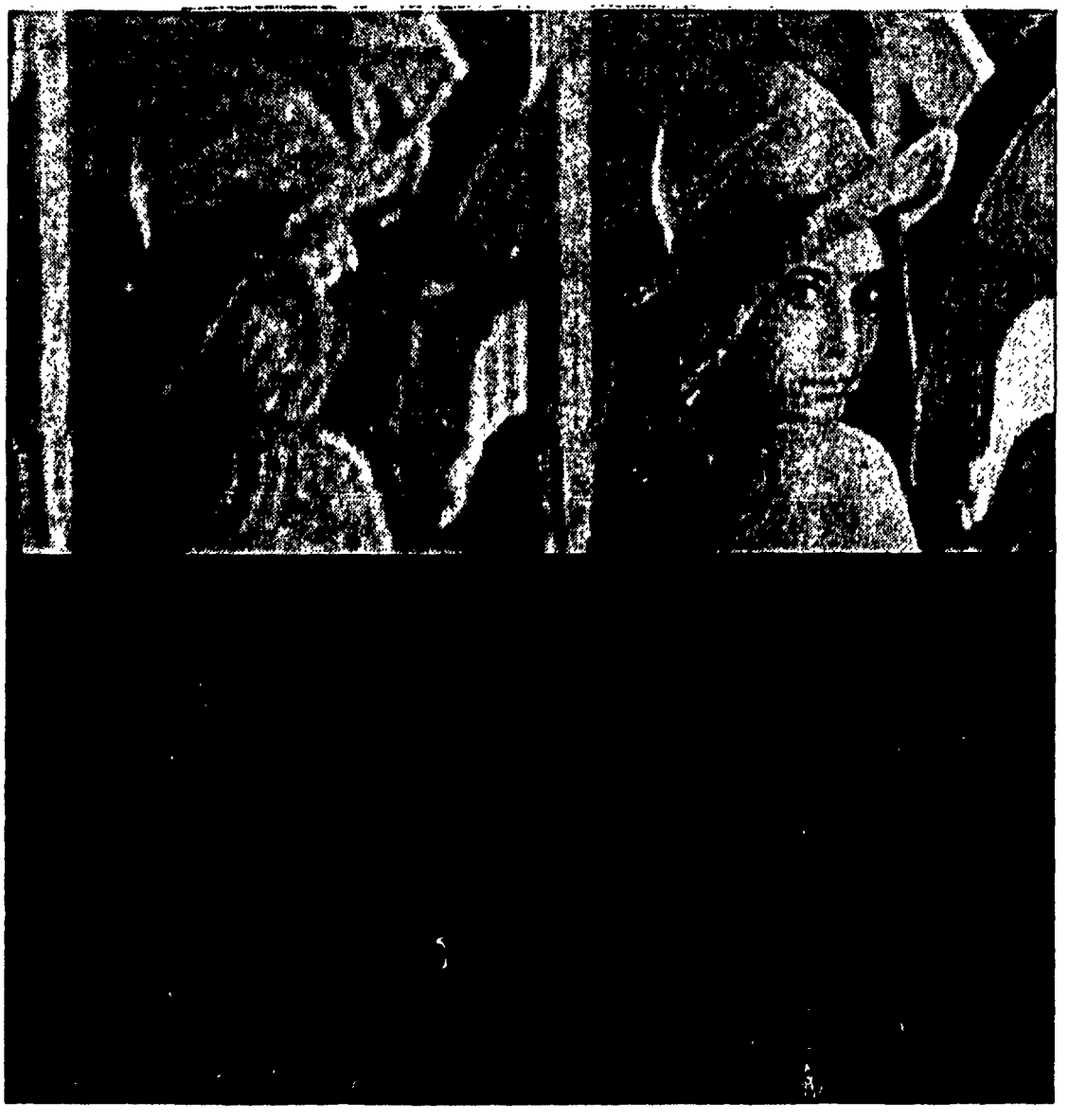


Figure 4.7: Application of Ideal Filters

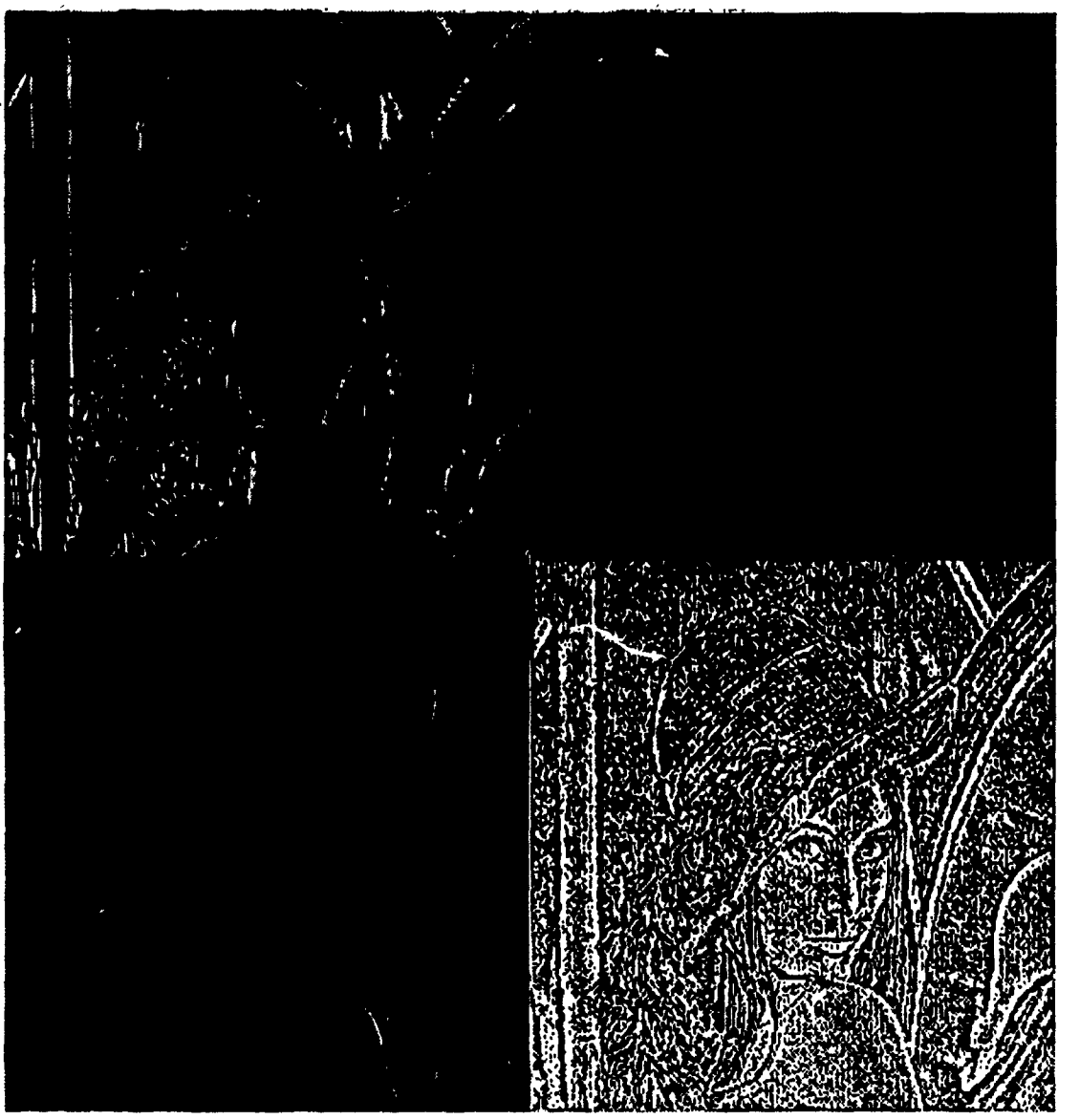


Figure 4.8: Edge Detectors

V. SUMMARY AND CONCLUSIONS

A. REVIEW OF THESIS

This thesis provides useful tools to make image processing easier and more flexible. Indeed, it makes image processing feasible in MATLAB. It supplies these tools in different environments in a consistent way in order to make the user interface identical for all environments. These tools are easy to use and flexible. We developed utilities for MATLAB, APL, C, and FORTRAN. Since MATLAB is a relatively new language, but is becoming a standard in signal processing and other areas, we have also provided a toolkit of specific functions for use with MATLAB. For the other languages, we have provided input/output functions. These functions hide the details of I/O from the user and provide a neat way to read and write images. If someone wants to process images with these languages, he or she can write programs that call these I/O functions.

For displaying images, we have provided two executable codes; one for PC compatibles and another for SUN SPARCstations. Both of these display programs are user friendly and easy to use. The workstation display program **show**, which runs under UNIX can also be used within PRO-MATLAB. This version is capable of displaying matrices (rather than image files) which makes it more useful for displaying results rapidly.

The MATLAB image processing toolkit consists of 10 m-files which implement some of the basic image processing functions. Since these are m-files, they work for PC-MATLAB, 386 MATLAB, or PRO-MATLAB. Examples of using these functions are presented in Chapter IV.

B. AREAS FOR FUTURE WORK

1. Displaying Color Images on SPARCstations

The display program for UNIX (SUN SPARCstations) is capable of displaying monochrome images, but for color images, it only displays image files in **rasterfile** form (with extension .ras). This format is not like the standard USC/SIPI color images where we have three separate files for red, green, and blue components. In **rasterfile** form, there is only one data file and three look-up tables for red, green, and blue components in the header part. These look-up tables are used to convert the single data file to three separate data arrays internally. The problem is: given three separate data files, find one data file and three look-up tables representing the same color image. If these look-up tables and one data file are found then a utility program **raw2ras** is provided to combine the data file and three look-up tables into a rasterfile.

2. Expanding the Toolkit

The basic toolkit can be expanded to cover more image processing functions and hence obtain more image processing capabilities within MATLAB. In addition, the basic toolkit is implemented entirely with m-files which makes it slow for some large matrix operations. If these functions are implemented with MEX files however, the speed will increase considerably.

APPENDIX A
Creating MEX Files

Programming in MATLAB is usually done via m-files. Another way is to call C or FORTRAN subroutines from MATLAB, as if they were built-in MATLAB functions. MATLAB-callable C and FORTRAN programs are referred to as MEX-files (MATLAB EXecutable files). MEX-files have several good applications:

- Large pre-existing FORTRAN and C codes can be called from MATLAB, without having to rewrite them as m-files.
- Bottleneck computations (usually for loops) are too slow in m-files. These can be re-coded in C or FORTRAN for efficiency. Speed improvements of up to a factor of 25 are possible.
- A/D cards, D/A cards, and other hardware can be accessed directly for data acquisition and control applications.
- Import/export of data can be accomplished (our functions for reading and saving images are an example of this).

The utilities for creating MEX files are discussed in Chapter 9 of the PC-MATLAB manual [Ref. 4]. The examples EIGS.C and VDPOL.C in the MEX directory are very useful in understanding the organization of MEX files. The I/O functions **readim, rim, saveim**, and **putim** were written in C and compiled with the CMEX utility. Those four functions explain how MEX files can be used for data import/export. Two other example functions **mexampl1** and **mexampl2** are provided in this appendix to explain how a variable in the MATLAB workspace can be reached, and how built-in functions, or m-files, or other MEX-files can be called from within a C program.

Important things to note about **mexampl1** are:

- The main function must be named **user_fcn()** instead of **main()**.
- Define a Matrix structure pointer for every variable to be reached.
- Check usage of function (number of left hand side and number of right hand side arguments need to be correct).
- Procedure **get_global()** with the variable name.
- Select the "pr member" of the Matrix structure which contains the actual data and assign it to the variable of the program.
- Print the results with function **mex_printf()**. Note that with the use of **mex_printf()**, we do not need to include <stdio.h>.

Important things to note about **mexampl2** are:

- Define **user_fcn()** instead of **main()**.
- Define Matrix structure pointers for other function calls (plhs[ ], and prhs[ ]).
- Call **matlab_fcn1()** to reach other m-files or MEX-files.
- The results will be in structure plhs.

mexampl1.c Program

Jun 13 11:20 1991 mexampl1.c Page 1

```
/*****************************************************************
 *                                                               *
 * this c program shows how to use procedures :                  *
 *     get_global()                                              *
 *     mex_printf()                                              *
 *                                                               *
 *  note :  get_global() is used to get the value of a constant  *
 *  (or a matrix) in user's workspace to a variable in this      *
 *   C-program.                                                  *
 *          mex_printf() can be used instead of printf() so      *
 *   you don't need to link the whole <stdio.h>                  *
 *                                                               *
 *****************************************************************/
#include <cmex.h>

user_fcn(nlhs,plhs,nrhs,prhs)
int nlhs,nrhs;
Matrix *plhs[],*prhs[];
{
        Matrix *ptr_to_ans;
        double *ans;

        if (nrhs!=0) mex_error("Must not be any input arguments");
        if (nlhs!=0) mex_error("Must not be any output arguments");

        ptr_to_ans=get_global("ans");

        ans=ptr_to_ans->pr;
        if(ans==0)   /* if NULL */
                mex_error("variable 'ans' does not exist yet");
        mex_printf("\nans = %f\n\n",*ans);

}
```

mexampl2.c Program

Jun 13 11:21 1991 mexampl2.c Page 1

```
/*************************************************************
 *                                                           *
 * this program shows how to call m-files, MEX-files or built-in*
 * functions from within an MEX-file.                        *
 * also read the example programs EIGS.C and VDPOL.C         *
 * in MEX directory.                                         *
 *                                                           *
 *************************************************************/

#include "cmex.h"

#define INP_1 prhs[0]
#define OUT_1 plhs[0]

user_fcn(nlhs, plhs, nrhs,prhs)
int nlhs, nrhs;
Matrix *plhs[], *prhs[];
{
    char *function_name1,*function_name2;

    if(nrhs!=1)     mex_error("must be one input");
    if(nlhs!=1)     mex_error("must be one output");

    function_name1="cos";
    function_name2="mexampl1";

    /* call built-in 'cosine' function */
    matlab_fcn(1,plhs,1,prhs,function_name1);

    /* call MEX-file 'mexampl1' */
    matlab_fcn(0,0,0,0,function_name2);

}
```

LIST OF REFERENCES

1. Gonzalez, Rafael C. and Wintz, Paul, *Digital Image Processing*, Addison-Wesley, 1987.

2. SUN Microsystems, Inc., "Pixrect Reference Manual 4.1," 1990.

3. Imaging Technology Inc., "ITEX PCplus PROGRAMMER'S MANUAL," 1987.

4. The MathWorks, Inc., "PC-MATLAB for MS-DOS Personal Computers," 1989.

5. Microsoft Corporation, "Microsoft C 5.0 Optimizing Compiler for the MS-DOS Operating System," 1984.

6. SUN Microsystems, Inc., "C Programmer's Guide," 1989.

7. APL2 Programming: "APL2 for the IBM PC User's Guide, Version 1.01," Mechanicsburg, PA, December 1988.

8. SUN Microsystems, Inc., "SUN FORTRAN Reference Manual," 1989.

INITIAL DISTRIBUTION LIST

	No. of Copies
1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3. Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4. Professor Charles W. Therrien, Code EC/Ti Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
5. Professor Jeffrey B. Burl, Code EC/Bl Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
6. Professor Roberto Cristi, Code EC/Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
7. Professor John Powers, Code EC/Po Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1

8. Professor Chin-Hwa Lee, Code EC/Le — 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

9. Professor Robert D. Strum, Code EC/St — 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

10. Professor J. Miller, Code EC/Mr — 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

11. Mr. Robert Limes, Code EC — 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

12. Erkan Aykaç — 2
Malkoc Mh 78.Sk No. 6
Gönen/BALIKESIR 10900
TURKEY